QSM368ZP-WF&SG368Z 系列
Linux&Ubuntu&OpenWrt
外设驱动开发指导

智能产品

版本：1.2

日期：2025-11-25

状态：受控文件

上海移远通信技术股份有限公司（以下简称“移远通信”）始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司
上海市闵行区田林路 1016 号科技绿洲 3 期（B 区）5 号楼　　邮编：200233
电话：+86 21 5108 6236　　邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：https://www.quectel.com.cn/contact。

如需技术支持或反馈我司技术文档中的问题，请随时登录网址：
https://www.quectel.com.cn/contact?tab=t 或发送邮件至：support@quectel.com。

前言

移远通信提供该文档内容以支持客户的产品设计。客户须按照文档中提供的规范、参数来设计产品。同时，您理解并同意，移远通信提供的参考设计仅作为示例。您同意在设计您目标产品时使用您独立的分析、评估和判断。在使用本文档所指导的任何硬软件或服务之前，请仔细阅读本声明。您在此承认并同意，尽管移远通信采取了商业范围内的合理努力来提供尽可能好的体验，但本文档和其所涉及服务是在“可用”基础上提供给您的。移远通信可在未事先通知的情况下，自行决定随时增加、修改或重述本文档。

使用和披露限制

许可协议

除非移远通信特别授权，否则我司所提供硬软件、材料和文档的接收方须对接收的内容保密，不得将其用于除本项目的实施与开展以外的任何其他目的。

版权声明

移远通信产品和本协议项下的第三方产品可能包含受移远通信或第三方材料、硬软件和文档版权保护的相关资料。除非事先得到书面同意，否则您不得获取、使用、向第三方披露我司所提供的文档和信息，或对此类受版权保护的资料进行复制、转载、抄袭、出版、展示、翻译、分发、合并、修改，或创造其衍生作品。移远通信或第三方对受版权保护的资料拥有专有权，不授予或转让任何专利、版权、商标或服务商标权的许可。为避免歧义，除了正常的非独家、免版税的产品使用许可，任何形式的购买都不可被视为授予许可。对于任何违反保密义务、未经授权使用或以其他非法形式恶意使用所述文档和信息的违法侵权行为，移远通信有权追究法律责任。

商标

除另行规定，本文档中的任何内容均不授予在广告、宣传或其他方面使用移远通信或第三方的任何商标、商号及名称，或其缩略语，或其仿冒品的权利。

第三方权利

您理解本文档可能涉及一个或多个属于第三方的硬软件和文档（“第三方材料”）。您对此类第三方材料的使用应受本文档的所有限制和义务约束。

移远通信针对第三方材料不做任何明示或暗示的保证或陈述，包括但不限于任何暗示或法定的适销性或特定用途的适用性、平静受益权、系统集成、信息准确性以及与许可技术或被许可人使用许可技术相关的不侵犯任何第三方知识产权的保证。本协议中的任何内容都不构成移远通信对任何移远通信产品或任何其他硬软件、设备、工具、信息或产品的开发、增强、修改、分销、营销、销售、提供销售或以其他方式维持生产的陈述或保证。此外，移远通信免除因交易过程、使用或贸易而产生的任何和所有保证。

隐私声明

为实现移远通信产品功能，特定设备数据将会上传至移远通信或第三方服务器（包括运营商、芯片供应商或您指定的服务器）。移远通信严格遵守相关法律法规，仅为实现产品功能之目的或在适用法律允许的情况下保留、使用、披露或以其他方式处理相关数据。当您与第三方进行数据交互前，请自行了解其隐私保护和数据安全政策。

免责声明

1) 移远通信不承担任何因未能遵守有关操作或设计规范而造成损害的责任。
2) 移远通信不承担因本文档中的任何因不准确、遗漏、或使用本文档中的信息而产生的任何责任。
3) 移远通信尽力确保开发中功能的完整性、准确性、及时性，但不排除上述功能错误或遗漏的可能。除非另有协议规定，否则移远通信对开发中功能的使用不做任何暗示或法定的保证。在适用法律允许的最大范围内，移远通信不对任何因使用开发中功能而遭受的损害承担责任，无论此类损害是否可以预见。
4) 移远通信对第三方网站及第三方资源的信息、内容、广告、商业报价、产品、服务和材料的可访问性、安全性、准确性、可用性、合法性和完整性不承担任何法律责任。

文档历史

修订记录

版本	日期	作者	变更表述
-	2023-06-14	Peter GAO/ Aaron YU	文档创建
1.0	2024-07-29	Peter GAO/ Aaron YU	受控版本
1.1	2024-09-10	Miloš Nedeljković	1. 更新文档名称，新增“OpenWrt”字样 2. 新增“OpenWrt”系统说明（第 1 章） 3. 更新 ADC 驱动文件路径和 DTS 节点配置参考文档（第 2.1 章和第 2.2 章） 4. 新增“OpenWrt”系统 **evtest** 命令测试备注说明（第 2.3 章） 5. 更新 UART 驱动文件路径（第 5.2.2 章） 6. 更新 GMAC 驱动代码路径（第 8.3 章）
1.2	2025-11-25	Damian Li	1. 添加 PDF 复制代码可能导致非预期换行的说明备注（第 1 章）。 2. 新增内核版本 Kernel 6.1。 3. ADC 功能：优化 *iio_channel_get()* 接口描述（第 2.4.1 章）。 4. SPI 功能：更新 SPI 主机及从设备节点配置备注（第 3.2.2 章）。 5. GPIO 功能：增加 GPIO 接口数量备注说明（第 7.1 章）。

目录

文档历史 3
目录 4
表格索引 6

1 引言 7

2 ADC 功能 8
2.1. 驱动文件 8
2.2. DTS 节点配置 8
2.3. ADC 使用 9
2.4. 常用接口 11
2.4.1. iio_channel_get 11
2.4.2. iio_read_channel_processed 12
2.5. ADC 节点使用示例 12

3 SPI 功能 13
3.1. SPI 配置 13
3.2. 设备树中 SPI 相关配置 14
3.2.1. 定义 SPI 别名 14
3.2.2. 配置 SPI 设备树 14
3.2.3. 配置 SPI pinctrl 15
3.2.4. 验证 SPI 主机配置 16
3.3. 内核中 SPI 相关文件 17
3.4. 内核中常用 SPI 接口函数 17

4 I2C 功能 18
4.1. I2C 配置 18
4.2. 设备树中 I2C 相关配置 19
4.2.1. 定义 I2C 别名 19
4.2.2. 配置 I2C 设备树 19
4.2.3. 配置 I2C pinctrl 20
4.2.4. 验证 I2C 适配器配置 21
4.3. 内核 I2C 接口函数 21

5 UART 功能 22
5.1. UART 介绍 22
5.1.1. UART 配置文件 23
5.2. UART 配置 24
5.2.1. 设备树中 UART 相关配置 24
5.2.2. UART 作为控制台 26
5.2.3. UART 测试方法 27
5.3. 用户空间中操作 UART 28

6 USB 功能 30
6.1. USB 驱动架构 30

6.1.1. 硬件架构 30
6.1.1.1. USB 控制器 30
6.1.1.2. USB PHY 30
6.1.1.3. USB 接口 31
6.1.2. 软件架构 31
6.1.2.1. USB 控制器驱动 31
6.1.2.2. USB PHY 驱动 32
6.1.2.3. USB 接口驱动 32
6.2. 设备树配置 33
6.3. USB 3.0 接口配置说明 35
6.3.1. USB 3.0 OTG 配置为 USB 2.0 Only 模式 35
6.3.2. USB 3.0 Host 配置为 USB 2.0 Only 模式 35
6.4. 切换 USB 功能 36

7 GPIO 功能 38
7.1. GPIO 介绍 38
7.2. 在内核中操作 GPIO 39
7.2.1. 在设备树中定义 GPIO 编号 39
7.2.2. 在内核驱动中配置 GPIO 方向 40
7.2.3. 在内核驱动中获取 GPIO 电平 41
7.2.4. 在内核驱动中修改 GPIO 输出电平 41
7.3. 用户空间 GPIO 操作示例 41

8 GMAC 功能 43
8.1. 概述 43
8.1.1. 以太网 PHY 芯片简介 43
8.1.2. 工作模式 43
8.2. 设备树配置 44
8.3. 配置以太网常见问题排查 46
8.3.1. DMA 初始化失败 46
8.3.2. PHY 初始化失败 46
8.3.3. 连接问题 47
8.3.4. 数据不通 47
8.3.4.1. TX 47
8.3.4.2. RX 48

9 附录 参考文档及术语缩写 49

表格索引

表 1：支持的 SPI 配置 13
表 2：支持的 I2C 配置 18
表 3：UART 接口复用 22
表 4：支持的 GPIO 配置 38
表 5：参考文档 49
表 6：术语缩写 49

1 引言

本文档主要介绍如何在移远通信 QSM368ZP-WF 设备和 SG368Z 系列模块上配置相关外设功能，如 ADC、SPI、I2C、UART、USB、GPIO 和 GMAC。

备注

1. 本文档适用于运行 Linux 操作系统的 QSM368ZP-WF 设备和运行 Linux 或 Ubuntu 或 OpenWrt 操作系统的 SG368Z 系列模块。
2. SG368Z 系列模块本身无法预装 OpenWrt 系统，仅提供 SDK 及开发指导供客户二次开发。若有问题，请联系移远通信技术支持。
3. 由于 PDF 格式特性，文档中部分代码块可能因页面宽度限制自动换行。直接从 PDF 复制代码到编辑器时，可能引入非预期的换行符（\n），导致代码无法正常运行。

   建议操作：

   若需从 PDF 复制，粘贴后请注意完成如下动作：
   - 手动删除复制粘贴后代码中的多余换行符。
   - 或通过代码编辑器的替换功能批量删除多余换行符（注意保留语义换行）。

2 ADC 功能

本章节主要介绍 QSM368ZP-WF 设备和 SG368Z 系列模块的 ADC 功能，并介绍如何使用 QSM368ZP-WF 设备和 SG368Z 系列模块提供的 ADC 接口。

QSM368ZP-WF 设备和 SG368Z 系列模块分别支持 5 路通用 ADC 通道，分别为 ADC2、ADC4~7。通道固定不变，用户无需配置，可以直接使用。

2.1. 驱动文件

ADC 驱动文件路径：

- Kernel 5.10 及之前：*kernel/drivers/iio/adc/rockchip_saradc.c*
- Kernel 6.1：*kernel-6.1/drivers/iio/adc/rockchip_saradc.c*

2.2. DTS 节点配置

DTS 文件位置：

- Kernel 5.10 及之前：
  - *kernel/arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi*
  - *kernel/arch/arm64/boot/dts/rockchip/rk3568.dtsi*
- Kernel 6.1：
  - *kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi*
  - *kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568.dtsi*
  - *kernel-6.1/arch/arm64/boot/dts/rockchip/rk356x.dtsi*

DTS 节点配置参考文档为：

- Kernel 5.10 及之前：*kernel/Documentation/devicetree/bindings/iio/adc/rockchip-saradc.yaml*
- Kernel 6.1：*kernel-6.1/Documentation/devicetree/bindings/iio/adc/rockchip-saradc.yaml*

配置示例如下：

```
        saradc: saradc@fe720000 {
                compatible = "rockchip,rk3568-saradc", "rockchip,rk3399-saradc";
                reg = <0x0 0xfe720000 0x0 0x100>;
                interrupts = <GIC_SPI 93 IRQ_TYPE_LEVEL_HIGH>;
                #io-channel-cells = <1>;
                clocks = <&cru CLK_SARADC>, <&cru PCLK_SARADC>;
                clock-names = "saradc", "apb_pclk";
                resets = <&cru SRST_P_SARADC>;
                reset-names = "saradc-apb";
                status = "disabled";
        };

 &saradc {
        status = "okay";
        vref-supply = <&vcca_1v8>;
};
```

重要参数说明如下：

参数	描述
interrupts	设置为 *GIC_SPI 93 IRQ_TYPE_LEVEL_HIGH*，表示数据转换完成，产生中断信号。
io-channel-cells	必须设置为 *1*，详见内核自带文档 *iio-bindings.txt*。
vref-supply	设置为*&vcca_1v8*，表示 SAR ADC 值对应的参考电压为 1.8 V，对应的 SAR ADC 值为 1024（参考电压需要根据具体的硬件环境设置）。 电压和 ADC 值成线性关系：电压 = ADC 值 × (1.8/1024)，单位：V。

2.3. ADC 使用

ADC 的使用依赖于 IIO 框架。在使用 IIO 框架时，需要初始化 *iio_dev* 结构体，具体步骤请参考 *rockchip_saradc_probe* 函数中的 *indio_dev* 结构体。完成结构体初始化后，调用驱动文件内部的 *devm_iio_device_register* 函数将 *indio_dev* 注册到 IIO 框架中，并等待“input”子系统注册。

以 *adc-key* 驱动为例，使用该驱动时，需初始化 *input_dev* 结构体，具体步骤请参考 *drivers/input/keyboard/adc-keys.c* 中的 *adc_keys_probe* 函数。完成结构体初始化后，调用 *input_register_device* 函数将 *input_dev* 注册进“input”子系统。

可以使用 **getevent** 命令测试 *adc-key* 驱动，测试时，假设 *adc-key* 为 event0，则使用 **getevent -s /dev/input/event0** 测试命令。调用关系如下：

```
adc_keys_poll -> iio_read_channel_processed -> iio_channel_read -> chan->indio_dev->info ->read_raw(rockchip_saradc_read_raw) ->iio_convert_raw_to_processed_unlocked
```

在上述调用关系中，*rockchip_saradc_read_raw()*是重要函数，以下是对该函数的分析：

1. 设置从上电到开始采样的间隔为 8 个 SCLK 周期。

```
writel_relaxed(8, info->regs + SARADC_DLY_PU_SOC);
```

2. 打开 SAR ADC，设置采样通道，使能中断并开始采样。

```
writel(SARADC_CTRL_POWER_CTRL | (chan->channel & SARADC_CTRL_CHN_MASK) | SARADC_CTRL_IRQ_ENABLE,info->regs + SARADC_CTRL);
```

3. 等待 SAR ADC 完成采样，并产生中断。

```
wait_for_completion_timeout(&info->completion, SARADC_TIMEOUT)
```

4. 完成采样后，将采样数据存放在 *val* 中。

```
*val = info->last_val;
```

调用 *iio_convert_raw_to_processed_unlocked* 将采样数据转换成对应的电压值。

中断处理需使用 *rockchip_saradc_isr* 函数，以下是对该函数的分析：

1. 保存数据，提供给上面的***步骤 4*** 使用。

```
info->last_val = readl_relaxed(info->regs + SARADC_DATA);
```

2. 清除中断，并关闭 SAR ADC。

```
writel_relaxed(0, info->regs + SARADC_CTRL);
```

整个采样过程为：

1. 使用 *rockchip_saradc_read_raw* 函数配置 SAR ADC；
2. 打开 SAR ADC；
3. 开始采样；
4. 等待中断；
5. 中断函数中清除中断；
6. 关闭 SAR ADC。

备注

1. 本章节通过 **getevent** 命令介绍 ADC 的函数调用过程。
2. 对于 Linux 系统，用户可以使用 **evtest** 命令测试。
3. 对于 Ubuntu 系统，用户使用 **evtest** 命令测试前，需自行安装，安装命令为：**sudo apt install evtest**。
4. 对于 OpenWrt 系统，用户使用 **evtest** 命令测试前，需自行安装，安装命令为：**opkg install evtest**。

2.4. 常用接口

内核使用的 IIO 子系统原生代码函数接口详情如下。

2.4.1. iio_channel_get

该函数用于根据通道名称获取对应 ADC 通道指针。

- **函数原型**

```
struct iio_channel *iio_channel_get(struct device *dev, const char *channel_name)
```

- **参数**

dev：
[In] 设备信息结构体。

channel_name：
[In] ADC 通道名称。

- **返回值**

对应 ADC 通道指针　　函数执行成功
对应 ADC 错误指针　　函数执行失败

2.4.2. iio_read_channel_processed

该函数用于查询对应 ADC 通道的电压、电流、温度或原始值。

- **函数原型**

```
int iio_read_channel_processed(struct iio_channel *chan, int *val)
```

- **参数**

chan：
[In] 查询的通道。

val：
[Out] 对应通道实际返回的电压、电流、温度或原始值。

- **返回值**

0　　函数执行成功
其他值　　函数执行失败

2.5. ADC 节点使用示例

QSM368ZP-WF 设备和 SG368Z 系列模块默认已创建 ADC 节点，用户可以使用 **cat** 命令读取相应 ADC 通道的原始值，如下所示（通道编号用*表示）：

```
cat /sys/bus/iio/devices/iio\:device0/in_voltage*_raw
```

以通道 0 为例，如果要读取通道 0 的 ADC 值，用户可以使用如下命令：

```
cat /sys/bus/iio/devices/iio\:device0/in_voltage0_raw
```

3 SPI 功能

本章节主要介绍如何在 QSM368ZP-WF 设备和 SG368Z 系列模块上配置 SPI，包括如何在设备树中以及内核中配置 SPI 和常用 SPI 接口函数介绍。

3.1. SPI 配置

QSM368ZP-WF 设备和 SG368Z 系列模块分别支持 4 个通用 SPI 控制器，特性如下：

1. 主模式和从模式；
2. 4、8 和 16 位串行数据传输；
3. 全双工和半双工模式传输。

表 1：支持的 SPI 配置

SPI 编号	复用情况	复用电压域
SPI0	M0、M1	M0：PMUIO2 M1：VCCIO5
SPI1	M0、M1	M0：VCCIO4 M1：VCCIO5
SPI2	M0、M1	M0：VCCIO4 M1：VCCIO5
SPI3	M0、M1	M0：VCCIO6 M1：VCCIO7

备注

1. 由于不同产品的应用具有灵活性，4 个 SPI 分别复用在几个不同的电源域，用后缀_M0 和_M1 区分不同复用位置。注意，_M0 和_M1 不能同时使用，分配时只能选择其中一组，不能在某些信号上选择 M0，在另一些信号上选择 M1。
2. QSM368ZP-WF 设备和 SG368Z 系列模块的 SPI 功能默认处于关闭状态，并且被复用为其他功能。如果需要使用 SPI 功能，需要关闭被复用的其他功能，再打开对应的 SPI 节点。

3.2. 设备树中 SPI 相关配置

3.2.1. 定义 SPI 别名

对于 QSM368ZP-WF 设备和 SG368Z 系列模块，设备树中所需的节点别名已默认配置：

- 在 Kernel 5.10 及之前内核版本中，位于 *kernel/arch/arm64/boot/dts/rockchip/rk3568.dtsi*；
- 在 Kernel 6.1 内核版本中，位于 *kernel-6.1/arch/arm64/boot/dts/rockchip/rk356x.dtsi*。

通常情况下无需额外修改；如有定制需求，可在对应文件的 aliases 节点中增补或调整。

```
    aliases {
        spi0 = &spi0;
        spi1 = &spi1;
        spi2 = &spi2;
        spi3 = &spi3;
    };
```

3.2.2. 配置 SPI 设备树

QSM368ZP-WF 设备和 SG368Z 系列模块的 SPI 主机配置需在以下设备树文件中完成，具体路径依据所用内核版本选择：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*

示例配置如下（以 *spi1* 为例）：

```
&spi1 {
        status = "okay";
        //assigned-clock-rates = <200000000>; //SPI 设备工作时钟值（默认不用配置）
        dma-names = "tx","rx";    //使能 DMA 模式，通讯长度少于 32 字节不建议用，置空赋值以
         禁用 DMA 模式，如"dma-names;";
        //rx-sample-delay-ns = <10>; //采用默认的采样延时配置（默认不用配置）
        //pinctrl 的配置，注意需要与实际硬件设计的接口对应，如果设计的是 M0，就配置为 M0，
设计为 M1 就配置为 M1
        pinctrl-names = "default", "high_speed";
        pinctrl-0 = <&spi0m0_cs0 &spi0m0_cs1 &spi0m0_pins>;
        pinctrl-1 = <&spi0m0_cs0 &spi0m0_cs1 &spi0m0_pins_hs>;
        spi_test@10 {
            compatible ="rockchip,spi_test_bus1_cs0";
            reg = <0>; //片选 0 或者 1
```

```
            spi-cpha; //设置 CPHA = 1，不配置则为 0
            spi-cpol; //设置 CPOL = 1，不配置则为 0
            spi-lsb-first; //先传输 lSB
            spi-max-frequency = <24000000>; //SPI 时钟输出的时钟频率，不超过 50 MHz
            status = "okay";
        };
};
```

备注

1. 以上为典型 SPI 主机及从设备节点配置。请根据实际硬件连接（如引脚复用模式 M0/M1）调整 pinctrl 配置。
2. *assigned-clock-rates* 和 *spi-max-frequency* 参数的配置说明：
   spi-max-frequency 是 SPI 的输出时钟频率，它是由 SPI 工作时钟 *assigned-clock-rates* 内部分频后得到的。由于内部分频至少是 2，因此它们之间的关系可以表示为 *assigned-clock-rates* ≥ 2 × *spi-max-frequency*。例如，如果需要达到 50 MHz 的 SPI I/O 速率，可以配置 *assigned-clock-rates* = *<100000000>*和 *spi-max-frequency* = *<50000000>*（内部分频为偶数分频）。即工作时钟为 100 MHz（PLL 分频到最接近但不超过 100 MHz 的值），然后再进行内部二分频，最终实现接近 50 MHz 的 I/O 速率。

3.2.3. 配置 SPI pinctrl

在如下文件中将对应的 GPIO 引脚复用为 SPI 功能：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568-pinctrl.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568-pinctrl.dtsi*

示例如下：

```
spi1 {
        /omit-if-no-ref/
        spi1m0_pins: spi1m0-pins {
            rockchip,pins =
                /* spi1_clkm0 */
                <2 RK_PB5 3 &pcfg_pull_none>,
                /* spi1_misom0 */
                <2 RK_PB6 3 &pcfg_pull_none>,
                /* spi1_mosim0 */
                <2 RK_PB7 4 &pcfg_pull_none>;
        };

        /omit-if-no-ref/
```

```
        spi1m0_cs0: spi1m0-cs0 {
            rockchip,pins =
                /* spi1_cs0m0 */
                <2 RK_PC0 4 &pcfg_pull_none>;
        };

        /omit-if-no-ref/
        spi1m0_cs1: spi1m0-cs1 {
            rockchip,pins =
                /* spi1_cs1m0 */
                <2 RK_PC6 3 &pcfg_pull_none>;
        };

        /omit-if-no-ref/
        spi1m1_pins: spi1m1-pins {
            rockchip,pins =
                /* spi1_clkm1 */
                <3 RK_PC3 3 &pcfg_pull_none>,
                /* spi1_misom1 */
                <3 RK_PC2 3 &pcfg_pull_none>,
                /* spi1_mosim1 */
                <3 RK_PC1 3 &pcfg_pull_none>;
        };

        /omit-if-no-ref/
        spi1m1_cs0: spi1m1-cs0 {
            rockchip,pins =
                /* spi1_cs0m1 */
                <3 RK_PA1 3 &pcfg_pull_none>;
        };
    };
```

备注

SPI 的 pinctrl 配置默认无需修改，已经根据功能配置好了 pinctrl 组，实际使用时，根据硬件设计选择对应的 pinctrl 组即可（选择 M0 或 M1，详见***第 3.2.2 章*** SPI 设备树配置中的 pinctrl）。

3.2.4. 验证 SPI 主机配置

SPI 主机配置完成后，执行 **ls /sys/class/spi_master/**验证配置。若存在 *spi*，则表示对应的 SPI 主机配置成功。

3.3. 内核中 SPI 相关文件

SPI 相关驱动在如下内核位置：

- SPI 驱动框架：*drivers/spi/spi.c*
- SPI 各接口实现代码：*drivers/spi/spi-rockchip.c*
- 创建 SPI 设备节点（供开发人员使用）：*drivers/spi/spidev.c*
- SPI 测试驱动（需要手动添加到 Makefile 编译）：*drivers/spi/spi-rockchip-test.c*
- SPI 测试工具（供开发人员使用）：*Documentation/spi/spidev_test.c*

3.4. 内核中常用 SPI 接口函数

内核中常用的 SPI 接口函数如下：

```
int spi_setup(struct spi_device *spi);
static inline int spi_write(struct spi_device *spi, const void *buf, size_t len);
static inline int spi_read(struct spi_device *spi, void *buf, size_t len);
static inline int spi_sync_transfer(struct spi_device *spi, struct spi_transfer *xfers, unsigned int num_xfers);
int spi_write_then_read(struct spi_device *spi, const void *txbuf, unsigned n_tx, void *rxbuf, unsigned n_rx);
```

4 I2C 功能

本章节以 I2C1 为例介绍如何在 QSM368ZP-WF 设备和 SG368Z 系列模块上配置 I2C，包括如何在设备树中配置 I2C 和常用 I2C 接口函数介绍。

4.1. I2C 配置

QSM368ZP-WF 设备和 SG368Z 系列模块分别支持 5 个 I2C 控制器，特性如下：

1. I2C 总线主模式；
2. 软件可编程时钟频率和传输速率高达 400 Kbps；
3. 7 位和 10 位寻址模式。

表 2：支持的 I2C 配置

I2C 编号	复用情况	复用电源域
I2C1	无	PMUIO2
I2C2	M0、M1	M0：PMUIO2 M1：VCCIO6
I2C3	M0、M1	M0：VCCIO1 M1：VCCIO5
I2C4	M0、M1	M0：VCCIO6 M1：VCCIO4
I2C5	M0、M1	M0：VCCIO5 M1：VCCIO7

备注

1. 由于不同产品的应用具有灵活性，5 个 I2C 分别复用在几个不同的电源域，用后缀_M0 和_M1 区分不同复用位置。注意，_M0 和_M1 不能同时使用，分配时只能选择其中一组，不能在某些信号上选择 M0，在另一些信号上选择 M1。
2. QSM368ZP-WF 设备和 SG368Z 系列模块的 I2C 功能默认处于关闭状态，并且被复用为其他功能。

如果需要使用 I2C 功能，需要关闭被复用的其他功能，再打开对应的 I2C 节点。

4.2. 设备树中 I2C 相关配置

4.2.1. 定义 I2C 别名

对于 QSM368ZP-WF 设备和 SG368Z 系列模块，设备树中所需的节点别名已默认配置：

- 在 Kernel 5.10 及之前内核版本中，位于 *kernel/arch/arm64/boot/dts/rockchip/rk3568.dtsi*；
- 在 Kernel 6.1 内核版本中，位于 *kernel-6.1/arch/arm64/boot/dts/rockchip/rk356x.dtsi*。

通常情况下无需额外修改；如有定制需求，可在对应文件的 aliases 节点中增补或调整。

```
 aliases {
            i2c0 = &i2c0;
            i2c1 = &i2c1;
            i2c2 = &i2c2;
            i2c3 = &i2c3;
            i2c4 = &i2c4;
            i2c5 = &i2c5;
}
```

4.2.2. 配置 I2C 设备树

QSM368ZP-WF 设备和 SG368Z 系列模块的 I2C 主机配置需在以下设备树文件中完成，具体路径依据所用内核版本选择：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*

示例配置如下（以 i2c1 为例）：

```
&i2c1 {
      status = "okay";
      pinctrl-names = "default";
      //对于有复用的 I2C，需要选择硬件实际使用的是 M0 还是 M1
      //例如 pinctrl-0 = <&i2c4m1_xfer>;
      pinctrl-0 = <&i2c1_xfer>;
      gt1x: gt1x@14 {
```

```
            compatible = "goodix,gt1x";
            reg = <0x14>;
        };
```

备注

1. 以上为典型 I2C 主机及从设备节点配置。请根据实际硬件连接（如引脚复用模式 M0/M1）调整 pinctrl 配置。
2. 如下三个参数默认无需配置：
   1) *clock-frequency*：默认时钟频率为 100 KHz，用户无需配置。如果需要使用其它的 I2C 时钟频率，则需要配置。如 400 KHz，则配置 *clock-frequency=<400000>*。最大可配置频率由 *i2c-scl-rising-time-ns* 决定；
   2) *i2c-scl-rising-time-ns*：SCL 上升沿时间，由硬件决定，可通过改变上拉电阻调节该时间。示波器可以用来测量实际的上升沿时间，例如，如果测得 SCL 上升沿时间为 365 纳秒，则配置为 *i2c-scl-rising-time-ns=<365>*（默认可以不配置，但必须保证当前的上升沿时间不能超过所配置频率下的 I2C 标准所定义的最大上升沿时间）。
   3) *i2c-scl-falling-time-ns*：SCL 下降沿时间，一般不变，等同于 *i2c-sda-falling-time-ns*（默认也可以不配置）。

4.2.3. 配置 I2C pinctrl

在如下文件中将对应的 GPIO 引脚复用为 I2C 功能：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568-pinctrl.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568-pinctrl.dtsi*

示例如下：

```
i2c1 {
    /omit-if-no-ref/
    i2c1_xfer: i2c1-xfer {
        rockchip,pins =
            /* i2c1_scl */
            <0 RK_PB3 1 &pcfg_pull_none_smt>,
            /* i2c1_sda */
            <0 RK_PB4 1 &pcfg_pull_none_smt>;
    };
};
```

备注

I2C 的 pinctrl 配置默认无需修改，已经根据功能配置好了 pinctrl 组，实际使用的时候，根据硬件设计选择对应的 pinctrl 组即可（选择 M0 或者 M1，详见***第 4.2.2 章*** I2C 设备树配置中的 pinctrl）。

4.2.4. 验证 I2C 适配器配置

配置好对应的 I2C 后，执行 **ls /sys/bus/i2c/devices** 验证配置。如果结果中包含对应的 I2C 名称表示该 I2C 配置成功。

4.3. 内核 I2C 接口函数

内核中常用的 I2C 接口函数如下所示：

```
s32 i2c_smbus_read_byte(const struct i2c_client *client)
s32 i2c_smbus_write_byte(const struct i2c_client *client, u8 value)
s32 i2c_smbus_write_byte_data(const struct i2c_client *client, u8 command, u8 value);
s32 i2c_smbus_read_byte_data(const struct i2c_client *client, u8 command);
s32 i2c_smbus_read_word_data(const struct i2c_client *client, u8 command)
s32 i2c_smbus_write_word_data(const struct i2c_client *client, u8 command, u16 value)
int   i2c_master_send(const struct i2c_client *client, const char *buf, int count);
int   i2c_master_recv(const struct i2c_client *client, char *buf, int count);
s32 i2c_smbus_read_block_data(const struct i2c_client *client, u8 command, u8 *values)
s32 i2c_smbus_write_block_data(const struct i2c_client *client, u8 command, u8 length, const u8 *values)
s32 i2c_smbus_read_i2c_block_data(const struct i2c_client *client, u8 command, u8 length, u8 *values)
s32 i2c_smbus_write_i2c_block_data(const struct i2c_client *client, u8 command, u8 length, const u8 *values)
int   i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msgs, int num);
```

5 UART 功能

本章节主要介绍 QSM368ZP-WF 设备和 SG368Z 系列模块的 UART 的配置和使能方法。

5.1. UART 介绍

QSM368ZP-WF 设备支持 3 组 UART 接口，1 组 1.8 V 和 2 组 3.3 V（接口电压）。SG368Z-WF 最多支持 8 组 UART 接口，分别是 UART2~UART9；SG368Z-AP 最多支持 10 组 UART 接口，分别是 UART0~UART9。UART 特性如下：

- 每个 UART 均包含两个 64 字节的 FIFO，用于数据接收和传输；
- 115.2 Kbps、460.8 Kbps、921.6 Kbps、1.5 Mbps、3 Mbps 和 4 Mbps 的波特率；
- 可编程波特率，支持非整数时钟分频器；
- 基于中断或基于 DMA 的模式；
- 5~8 位宽度的数据传输。

备注

1. QSM368ZP-WF 设备提供 RS232 和 RS485 接口各 1 个，共 2 个，使用时，均需用户自己贴片。
2. QSM368ZP-WF 设备的 UART 接口引脚已固定，因此引脚部分配置无法修改或复用成其他功能。

QSM368ZP-WF 设备和 SG368Z 系列模块的 UART 控制器，除 UART0 外，其他接口都有多组接口引脚可以选择。UART 接口复用如下表所示：

表 3：UART 接口复用

UART 编号	接口复用	所属电源域
UART0	无接口复用	PMUIO2
UART1	M0、M1	M0：VCCIO4 M1：VCCIO6

UART2	M0、M1	M0：PMUIO2 M1：VCCIO3
UART3	M0、M1	M0：VCCIO1 M1：VCCIO5
UART4	M0、M1	M0：VCCIO1 M1：VCCIO5
UART5	M0、M1	M0：VCCIO3 M1：VCCIO5
UART6	M0、M1	M0：VCCIO4 M1：VCCIO3
UART7	M0、M1、M2	M0：VCCIO4 M1：VCCIO5 M2：VCCIO6
UART8	M0、M1	M0：VCCIO4 M1：VCCIO5
UART9	M0、M1、M2	M0：VCCIO4 M1：VCCIO7 M2：VCCIO6

备注

1. 由于不同产品的应用具有灵活性，10 组 UART 分别复用在几个不同的电源域，用后缀_M0 和_M1 区分不同复用位置。注意，_M0 和_M1 不能同时使用，分配时只能选择其中一组，不能在某些信号上选择 M0，在另一些信号上选择 M1。
2. UART2_M0 默认是 QSM368ZP-WF 设备和 SG368Z 系列模块的调试接口，因此，打开调试接口功能后，需要关闭 UART2 串口功能，默认波特率为 115.2 Kbps。

5.1.1. UART 配置文件

UART 配置文件及其路径如下：

- 串口驱动核心文件：*drivers/tty/serial/8250/8250_core.c*
- Synopsis DesignWare 串口驱动文件：*drivers/tty/serial/8250/8250_dw.c*
- 串口 DMA 驱动文件：*drivers/tty/serial/8250/8250_dma.c*
- 串口端口操作文件：*drivers/tty/serial/8250/8250_port.c*
- 串口 Early Console 驱动文件：*drivers/tty/serial/8250/8250_early.c*

5.2. UART 配置

5.2.1. 设备树中 UART 相关配置

UART 的原始节点在如下设备树文件中定义：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk356x.dtsi*

示例如下：

```
uart1: serial@fe650000 {
compatible = "rockchip,rk3568-uart", "snps,dw-apb-uart";
reg = <0x0 0xfe650000 0x0 0x100>;
interrupts = <GIC_SPI 117 IRQ_TYPE_LEVEL_HIGH>;
clocks = <&cru SCLK_UART1>, <&cru PCLK_UART1>;
clock-names = "baudclk", "apb_pclk";
reg-shift = <2>;
reg-io-width = <4>;
dmas = <&dmac0 2>, <&dmac0 3>;
dma-names = "tx", "rx";
pinctrl-names = "default";
pinctrl-0 = <&uart1m0_xfer>;
status = "disabled";
};
```

用户可修改以下参数：

参数	描述
dma-names	DMA 通道名称。 *"tx"*：打开 TX DMA *"rx"*：打开 RX DMA *"!tx"*：关闭 TX DMA *"!rx"*：关闭 RX DMA
pinctrl-0	引脚定义。 *&uart1m0_xfer*：配置 TX 和 RX 引脚为 iomux group 0 *&uart1m1_xfer*：配置 TX 和 RX 引脚为 iomux group 1 *&uart1m0_ctsn*：配置硬件自动流控 CTS 引脚为 iomux group 0 *&uart1m1_ctsn*：配置硬件自动流控 CTS 引脚为 iomux group 1 *&uart1m0_rtsn*：配置硬件自动流控 RTS 引脚为 iomux group 0 *&uart1m1_rtsn*：配置硬件自动流控 RTS 引脚为 iomux group 1
status	节点的状态。

	"okay"：打开 *"disabled"*：关闭

以上参数在如下设备树文件中配置：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*

配置示例如下：

```
&uart1 {
dma-names = "tx", "rx";
pinctrl-names = "default";
pinctrl-0 = <&uart1m0_xfer &uart1m0_ctsn &uart1m0_rtsn>;
status = "okay";
};
```

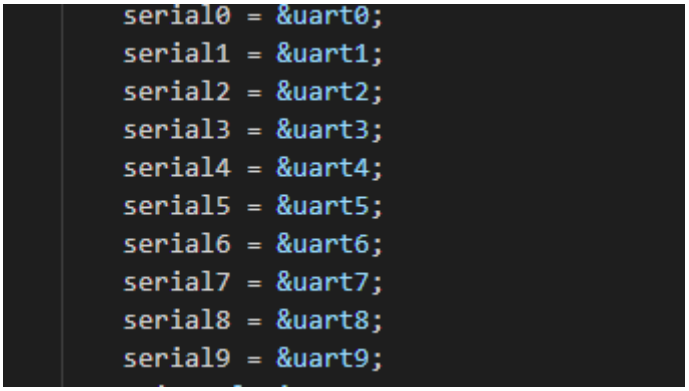
备注

参数 *pinctrl-0* 中对于硬件自动流控 CTS 和 RTS 引脚的操作仅仅是配置引脚为 iomux，实际使能硬件自动流控 CTS 和 RTS 引脚的操作在***第 5.1.1 章***的 UART 配置文件中完成，如果不需要使用硬件自动流控，可以删除硬件自动流控 CTS 和 RTS 引脚的 iomux 配置。

使能 UART 后，在系统启动的 log 中可以看到以下对应的打印，表示设备正常注册：

```
fe650000.serial: ttyS1 at MMIO 0xfe650000 (irq = 67, base_baud = 1500000) is a
16550A
```

普通串口设备会根据 *rk3568.dtsi*（Kernel 5.10 及之前）或在 *rk356x.dtsi*（Kernel 6.1）文件中的别名来对串口进行编号，对应注册成 *ttySx* 设备。文件中的别名如下：



例如，如果需要把 UART3 注册成 *ttyS1*，可以进行如下修改：

```
aliases {
serial0 = &uart0;
serial1 = &uart3;
serial2 = &uart2;
serial3 = &uart1;
}
```

5.2.2. UART 作为控制台

UART 作为控制台（调试串口），使用 *fiq_debugger* 流程。一般将 UART2 配置为 *ttyFIQ0* 设备。不同 Kernel 版本的驱动文件路径不同，如下所示：

- Kernel 5.10：*drivers/soc/rockchip/fiq_debugger/fiq_debugger.c*
- Kernel 4.4：*drivers/soc/rockchip/rk_fiq_debugger.c*
- Kernel 6.1：*drivers/soc/rockchip/fiq_debugger/fiq_debugger.c*

设备树节点：

- Kernel 5.10 及以前：*kernel/arch/arm64/boot/dts/rockchip/rk3568-android.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568-android.dtsi*

```
chosen: chosen {
bootargs = "earlycon=uart8250,mmio32,0xfe660000 console=ttyFIQ0";
};
fiq-debugger {
compatible = "rockchip,fiq-debugger";
rockchip,serial-id = <2>;
rockchip,wake-irq = <0>;
/* If enable uart uses irq instead of fiq */
rockchip,irq-mode-enable = <1>;
rockchip,baudrate = <1500000>; /* Only 115200 and 1500000 */
interrupts = <GIC_SPI 252 IRQ_TYPE_LEVEL_LOW>;
pinctrl-names = "default";
pinctrl-0 = <&uart2m0_xfer>;
status = "okay";
};
&uart2 {
status = "disabled";
};
```

重要参数说明如下：

参数	描述
rockchip,serial-id	使用的 UART 编号。 通过将 *serial-id* 修改为不同的 UART 编号，*fiq_debugger* 设备也会相应地注册成 *ttyFIQ0* 设备。
rockchip,irq-mode-enable	配置中断方式。 *1* IRQ 中断 *0* FIQ 中断
interrupts	配置的辅助中断，保持默认即可。

备注

由于 *fiq_debugger* 和普通串口互斥，在使能 *fiq_debugger* 节点后必须禁用对应的普通串口 UART 节点。

5.2.3. UART 测试方法

测试发送的命令如下（*send_0x55* 和 *send_00_ff* 为发送的文件）：

```
/data/ts_uart.uart s ./data/send_0x55 1500000 0 0 0 /dev/ttyS1
./data/ts_uart.uart s ./data/send_00_ff 1500000 0 0 0 /dev/ttyS1
```

可以通过将 ADB 与 PC 端连接，使用 PC 端串口调试工具查看发送是否成功。

测试接收的命令如下（*receive_0x55* 为接收的文件）：

```
./data/ts_uart.uart r ./data/receive_0x55 1500000 0 0 0 /dev/ttyS1
```

可以使用 PC 端串口调试工具发送数据，测试程序将自动检测，检测到 *0x55* 表示接收正确，检测到其它字符将打印 16 进制 ASCII 码值，可对照查询接收是否正确。

测试内部自发自收的命令如下，此测试是 UART 控制器 IP 内部的自环测试。

```
./data/ts_uart.uart m ./data/send_00_ff 1500000 0 0 0 /dev/ttyS1
```

按住“**Ctrl + C**”停止测试，结束 log 如下，比较发送和接收的数据是否一致。

```
Sending data from file to port...
send:1172, receive:1172 total:1172      //收发数据一致，测试成功
send:3441, receive:3537 total:3441      //收发数据不一致，测试失败
```

5.3. 用户空间中操作 UART

在用户空间中操作 UART 读写数据的示例如下：

```
#define SPEED B115200
#define BUF_SIZE (4 * 1024)

static int open_port(char *node)
{
	int fd;
	struct termios options;
	fd = open(node, O_RDWR | O_NOCTTY | O_NDELAY);
	if (fd < 0) {
		printf("Could not open %s/n", node);
		return -1;
	}
	if (tcflush(fd, TCIOFLUSH) < 0) {
		printf("Could not flush uart port");
		return -1;
	}
	if (tcgetattr(fd, &options) < 0) {
		printf("Can't get port settings /n");
		return -1;
	}
	options.c_cc[VTIME] = 0;
	options.c_cc[VMIN] = 1;
	options.c_cflag &= ~CSIZE;
	options.c_cflag |= (CS8 | CLOCAL | CREAD);
	options.c_iflag = IGNPAR;
	options.c_oflag = 0;
	options.c_lflag = 0;
	cfsetospeed(&options, B115200);
	cfsetispeed(&options, B115200);
	if (tcsetattr(fd, TCSANOW, &options) < 0) {
		printf("Can't set port setting /n");
		return -1;
	}
	fcntl(fd, F_SETFL, 0);
	return fd;
}
void clear_buf(unsigned char *buf)
{
	int i;
```

```
    for(i = 0; i < BUF_SIZE; i++)
        buf[i] = 0x00;
}
int main(int argc, char *argv[])
{
    int fd;
    unsigned char buf[BUF_SIZE];
    if(argc != 2) {
        printf("usage:%s /dev/ttyS1/n", argv[0]);
        return 0;
    }
    fd = open_port(argv[1]);
    if(fd < 0)
        return -1;
    while(1) {
        read(fd, buf, BUF_SIZE);
        write(fd, buf, strlen(buf));
        clear_buf(buf);
    }
    close(fd);
    return 0;
}
```

6 USB 功能

本章节主要介绍 QSM368ZP-WF 设备和 SG368Z 系列模块上 USB 驱动的软硬件架构、常用的 USB 配置方法以及如何切换 USB 功能。

6.1. USB 驱动架构

6.1.1. 硬件架构

从硬件角度来看，USB 包括三个部分：USB 控制器、USB PHY 和 USB 接口。

备注

有关详细信息，请参考*文档 [1]*和*文档 [2]*。

6.1.1.1. USB 控制器

USB 控制器是 USB 协议的逻辑电路，固化在芯片内部，实现 USB 协议处理。

QSM368ZP-WF 设备和 SG368Z 系列模块分别支持 4 个不同的 USB 控制器：2 个 USB 2.0 Host、1 个 USB 3.0 Host 和 1 个 USB 3.0 OTG。QSM368ZP-WF 设备的 USB 硬件接口已经设计好，其中的 USB 3.0 OTG 口被拆分为 Micro USB 2.0 接口和 USB 3.0 Type-A 接口，本质上属于一个控制器，所以两者工作模式需相同，同主或同从。

USB 2.0 支持 EHCI 和 OHCI，USB 3.0 支持 DWC3 和 xHCI。

6.1.1.2. USB PHY

USB PHY 用于完成 USB 信号的数模转换。QSM368ZP-WF 设备和 SG368Z 系列模块 USB PHY 设计较为复杂，注意软件和硬件的配置要一致。USB 控制器和 USB PHY 之间的关系如下：

- USB 2.0 Host_2 控制器与 USB 2.0 Host_3 控制器分别使用 USB 2.0 Comb PHY_1 的 port0 和 port1 端口（此 PHY 有 2 个端口）；

- USB 3.0 OTG 控制器与 SATA_0 控制器复用 USB3/SATA Combo PHY_0；
- USB 3.0 Host_1 控制器与 SATA_1/QSGMII 控制器复用 USB3/SATA/QSGMII Combo PHY_1；
- USB 3.0 OTG 控制器与 USB 3.0 Host_1 控制器分别使用 USB 2.0 Comb PHY_0 的 port0 和 port1（兼容 USB 2.0）。

备注

1. 当 USB 3.0 Host_1 设计为 USB 3.0 的接口时，SATA_1 功能需要在设备树中关闭。
2. 当 USB 3.0 OTG 设计为 USB 3.0 的接口时，SATA_0 功能需要在设备树中关闭。
3. 当 USB 3.0 弃用 USB 3.0，设计为 USB 2.0 的接口时，USB 3.0 的通道可以相应的配置为其他功能（SATA 或 QSGMII）。

6.1.1.3. USB 接口

QSM368ZP-WF 设备和 SG368Z 系列模块支持 Type-C、Type-A 和 Micro-B 三种 USB 接口。

- Type-C：一般用作 USB OTG 接口。QSM368ZP-WF 设备和 SG368Z 系列模块需要外挂 CC logic 芯片才能实现完整的 Type-C 功能，向下兼容 USB 2.0；
- Type-A：一般用作 USB Host 接口，用于接 U 盘、鼠标等设备；
- Micro-B：一般用作接 USB 2.0 OTG 接口，可以通过 USB_ID 切换主从，无需额外的芯片。

6.1.2. 软件架构

USB 软件架构依赖硬件部分。USB 驱动也包括三个部分：USB 控制器驱动、USB PHY 驱动和 USB 接口驱动（CC logic 驱动）。

6.1.2.1. USB 控制器驱动

USB 控制器驱动直接驱动 USB 控制器。QSM368ZP-WF 设备和 SG368Z 系列模块支持两种不同的 USB 控制器驱动，分别为 USB 2.0 和 USB 3.0，对应的源码路径如下：

1. USB 3.0 源码路径：
   - Kernel 5.10 及之前：
     - *kernel/drivers/usb/dwc3/dwc3-of-simple.c*

- *kernel/drivers/usb/dwc3/core.c*

- Kernel 6.1：
  - *kernel-6.1/drivers/usb/dwc3/dwc3-of-simple.c*
  - *kernel-6.1/drivers/usb/dwc3/core.c*

2. USB 2.0 源码路径：

- Kernel 5.10 及之前：
  - *kernel/drivers/usb/host/ehci-platform.c*
  - *kernel/drivers/usb/host/ohci-platform.c*
- Kernel 6.1：
  - *kernel-6.1/drivers/usb/host/ehci-platform.c*
  - *kernel-6.1/drivers/usb/host/ohci-platform.c*

6.1.2.2. USB PHY 驱动

- Kernel 5.10 及之前：
  1. USB 2.0 PHY 源码路径：*kernel/drivers/phy/rockchip/phy-rockchip-inno-usb2.c*
  2. USB 3.0 PHY 源码路径：*kernel/drivers/phy/rockchip/phy-rockchip-naneng-combphy.c*
- Kernel 6.1：
  1. USB 2.0 PHY 源码路径：*kernel-6.1/drivers/phy/rockchip/phy-rockchip-inno-usb2.c*
  2. USB 3.0 PHY 源码路径：*kernel-6.1/drivers/phy/rockchip/phy-rockchip-naneng-combphy.c*

6.1.2.3. USB 接口驱动

USB 接口驱动源码路径：

- Kernel 5.10 及之前：*kernel/drivers/misc/pericom_i2c_30216c.c*

- Kernel 6.1：*kernel-6.1/drivers/misc/pericom_i2c_30216c.c*

6.2. 设备树配置

USB 的设备树源文件路径为：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk356x.dtsi*

OTG 的控制器设备节点默认配置如下图所示（用户无需修改）：

```
usbdrd30: usbdrd {
    compatible = "rockchip,rk3568-dwc3", "rockchip,rk3399-dwc3";
    clocks = <&cru CLK_USB3OTG0_REF>, <&cru CLK_USB3OTG0_SUSPEND>,
         <&cru ACLK_USB3OTG0>, <&cru PCLK_PIPE>;
    clock-names = "ref_clk", "suspend_clk",
              "bus_clk", "pipe_clk";
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;
    status = "disabled";

    usbdrd_dwc3: dwc3@fcc00000 {
        compatible = "snps,dwc3";
        reg = <0x0 0xfcc00000 0x0 0x400000>;
        interrupts = <GIC_SPI 169 IRQ_TYPE_LEVEL_HIGH>;
        dr_mode = "otg";
        phys = <&u2phy0_otg>, <&combphy0_us PHY_TYPE_USB3>;
        phy-names = "usb2-phy", "usb3-phy";
        phy_type = "utmi_wide";
        power-domains = <&power RK3568_PD_PIPE>;
        resets = <&cru SRST_USB3OTG0>;
        reset-names = "usb3-otg";
        snps,dis_enblslpm_quirk;
        snps,dis-u1u2-quirk;
        snps,dis-u2-freeclk-exists-quirk;
        snps,dis-del-phy-power-chg-quirk;
        snps,dis-tx-ipgap-linecheck-quirk;
        snps,dis_rxdet_inp3_quirk;
        snps,xhci-trb-ent-quirk;
        quirk-skip-phy-init;
        status = "disabled";
    };
```

USB 的硬件引脚配置文件为：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568-usb.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/usb/rk3568-usb.dtsi*

默认配置如下所示（若硬件无更改，用户无需修改此文件）：

```
vcc5v0_host: vcc5v0-host-regulator {
    compatible = "regulator-fixed";
    regulator-name = "vcc5v0_host";
    regulator-boot-on;
    regulator-always-on;
    regulator-min-microvolt = <5000000>;
    regulator-max-microvolt = <5000000>;
    enable-active-high;
    gpio = <&gpio0 RK_PA6 GPIO_ACTIVE_HIGH>;
    vin-supply = <&vcc5v0_usb>;
    pinctrl-names = "default";
    pinctrl-0 = <&vcc5v0_host_en>;
};

vcc5v0_otg: vcc5v0-otg-regulator {
    compatible = "regulator-fixed";
    regulator-name = "vcc5v0_otg";
    regulator-min-microvolt = <5000000>;
    regulator-max-microvolt = <5000000>;
    enable-active-high;
    gpio = <&gpio3 RK_PC1 GPIO_ACTIVE_HIGH>;
    vin-supply = <&vcc5v0_usb>;
    pinctrl-names = "default";
    pinctrl-0 = <&vcc5v0_otg_en>;
};
```

```
usb {   pinctrl配置
    vcc5v0_usb_en: vcc5v0-usb-en {
        rockchip,pins = <3 RK_PD1 RK_FUNC_GPIO &pcfg_pull_none>;
    };
    vcc5v0_host_en: vcc5v0-host-en {
        rockchip,pins = <0 RK_PA6 RK_FUNC_GPIO &pcfg_pull_none>;
    };

    vcc5v0_otg_en: vcc5v0-otg-en {
        rockchip,pins = <3 RK_PC1 RK_FUNC_GPIO &pcfg_pull_none>;
    };
};
```

配置说明如下：

1. 默认 VBUS 电源由对应的 GPIO 控制上电，因此，需要配置 GPIO 对应引脚的状态和功能。若硬件设计没有修改，则无需修改引脚的状态和功能。

2. *vcc5v0_otg* 是 OTG 的 VBUS 电源配置，*vcc5v0_host* 是 USB HOST 的电源配置。

3. *vcc5v0_usb* 是 OTG 和 Host 的上一级电源，如果该电源无 GPIO 控制，可以省略 GPIO 和 pinctrl 相关配置，只配置单独的节点用来表示电源的上下级关系。

4. pinctrl 配置用于设置对应电源 GPIO 的初始化状态。

6.3. USB 3.0 接口配置说明

QSM368ZP-WF 设备和 SG368Z 系列模块的 USB 3.0 接口可配置为纯 USB 2.0 接口使用，而此时，可将 USB 3.0 的引脚用于其他功能，例如 SATA 或 QSGMII。

6.3.1. USB 3.0 OTG 配置为 USB 2.0 Only 模式

USB 3.0 OTG 接口的 USB 3.0 PHY 与 SATA_0 控制器共用 Comb PHY_0，当配置 Comb PHY_0 用于 SATA_0 时，需要将 USB 3.0 OTG 配置在 USB 2.0 Only 模式下工作。以下是相应的 DTS 配置示例（无需改动控制器和 PHY 驱动）：

```
&usbdrd_dwc3 {
phys = <&u2phy0_otg>;                    //配置 phys 属性只引用 USB 2.0 PHY 节点
phy-names = "usb2-phy";
extcon = <&usb2phy0>;
maximum-speed = "high-speed";            //配置 DWC3 控制器最高支持 high-speed
snps,dis_u2_susphy_quirk;                //配置 DWC3 控制器禁用 USB 2.0 PHY 自动挂起的功能
};
&combphy0_us {
rockchip,dis-u3otg0-port;                //配置 DWC3_0 控制器最高支持 high-speed
status = "okay";
};
```

6.3.2. USB 3.0 Host 配置为 USB 2.0 Only 模式

USB 3.0 Host 的 USB 3.0 PHY 与 SATA_1/QSGMII 控制器共用 Comb PHY_1，当配置 Comb PHY_1 用于 SATA_1 或 QSGMII 控制器时，需要将 USB 3.0 Host 配置在 USB 2.0 Only 模式下工作。以下是相应的 DTS 配置示例（无需改动控制器和 PHY 驱动）：

```
&usbhost_dwc3 {
phys = <&u2phy0_host>;                  //配置 phys 属性只引用 USB 2.0 PHY 节点
phy-names = "usb2-phy";
maximum-speed = "high-speed";           //配置 DWC3 控制器最高支持 high-speed
snps,dis_u2_susphy_quirk;               //配置 DWC3 控制器禁用 USB 2.0 PHY 自动挂起的功能
status = "okay";
};
&combphy1_usq {
rockchip,dis-u3otg1-port;               //配置 DWC3_1 控制器最高支持 high-speed
status = "okay";
};
```

6.4. 切换 USB 功能

USB 功能配置文件为*/etc/init.d/.usb_config*，在系统运行状态下修改 USB 功能，可以修改*/tmp/.usb_config*，并运行**/etc/init.d/usbdevice.sh restart** 重置 USB 功能。

目前 USB Gadget 自动配置支持以下选项（在*.usb_config* 文件中配置）：

```
usb_adb_en
usb_uac1_en
usb_uac2_en
usb_rndis_en
usb_mtp_en
usb_ums_en
usb_acm_en
usb_uvc_en
```

/etc/init.d/.usb_config 和*/tmp/.usb_config* 说明：

/etc/init.d/.usb_config 为开机默认配置，因为系统可能为只读系统，该部分文件可能无法修改。因此特殊原因，运行状态下不建议修改该文件。

/tmp/.usb_config 是*/etc/init.d/.usb_config* 的复制文件，开机后，*usbdevice* 会将*/etc/init.d/.usb_config* 复制一份到*/tmp* 目录下，*/tmp* 目录为 RAM 空间，可以随时修改。如果需要在系统运行状态下临时修改 USB 功能，可以修改该文件，并用**/etc/init.d/usbdevice.sh restart** 重置 USB 功能。

示例如下：

```
echo usb_adb_en > /tmp/.usb_config
/etc/init.d/usbdevice.sh restart
```

备注

1. 部分功能依赖于上层的服务，比如 ADB 功能，依赖于上层的 *adbd* 服务；
2. 通过执行 **echo** 命令打开的功能是临时生效的，设备重启后会失效。要使其永久生效，需要修改 *buildroot/board/rockchip/rk356x/fs-overlay/etc/init.d/.usb_config* 文件。修改该文件时，需注意将 *usb_adb_en* 放在最后一个。

7 GPIO 功能

本章节介绍了如何在 QSM368ZP-WF 设备和 SG368Z 系列模块上配置 GPIO，包括设备树配置和用户空间的操作示例。

7.1. GPIO 介绍

SG368Z-AP 模块支持 128 个 GPIO 接口，而 QSM368ZP-WF 设备和 SG368Z-WF 分别支持 107 个 GPIO。支持的 GPIO 配置信息如下表所示：

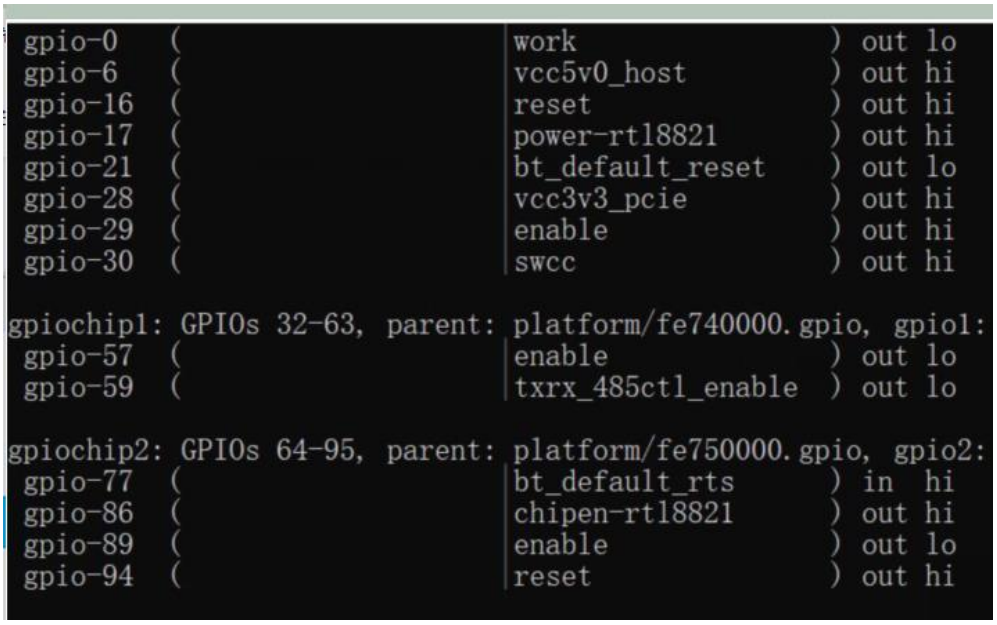
表 4：支持的 GPIO 配置

设备	QSM368ZP-WF	SG368Z-AP	SG368Z-WF
支持的 GPIO 接口数量	107	128	107
输入配置取值及说明	*pcfg_pull_down*：下拉 *pcfg_pull_up*：上拉 *pcfg_pull_none*：无上下拉		
输出配置	可编程驱动电流		
顶级模式多路复用控制器	用于便捷地编程多路 GPIO		

备注

1. QSM368ZP-WF 设备和 SG368Z 系列模块的 GPIO 名称采用 *gpiox_yz* 的编号方法。其中 *x* 代表数字 0 ~ 4，表示有 5 个 GPIO 控制器；*y* 代表字母 A ~ D，表示每个控制器分为 4 组；*z* 代表数字 0 ~ 7，表示每组有 8 个引脚。例如：gpio0_C4 表示 GPIO0 控制器的第 3 组的第 4 个引脚。
2. 部分 GPIO 接口因内部复用或封装限制无法对外使用，实际可用数量以上表所列为准。有关各模块的 GPIO 接口详情，请参考***文档 [2]***。

系统开机时，可在系统中看到下图内容：



每个控制器有 32 个引脚，而每组有 8 个引脚。因此，硬件设计上 GPIO 的编号名称和内核显示的 GPIO 号之间具有以下转换关系：

gpio0_C0：0 × 32 + 2 × 8 + 0 = 16 --->gpio16
gpio2_A1：2 × 32 + 0 × 8 + 1 = 65 --->gpio65
gpio2_B5：2 × 32 + 1 × 8 + 5 = 77 --->gpio77

7.2. 在内核中操作 GPIO

7.2.1. 在设备树中定义 GPIO 编号

在如下设备树文件中定义 GPIO 编号：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*

示例如下：

```
vcc_5Gpower:vcc_5Gpower {
        compatible = "regulator-fixed";
        regulator-name = "out5G_power";
        enable-active-high;
        regulator-always-on;
        pinctrl-names = "default";
```

```
            pinctrl-0 = <&gpio_5G_power>;
            gpio = <&gpio3 RK_PB3 GPIO_ACTIVE_HIGH>;
            status = "okay";
    };
&pinctrl {
            ……
      gpio-5Gpower {
            gpio_5G_power:gpio-5G-power {
                  rockchip,pins = <3 RK_PB3 RK_FUNC_GPIO &pcfg_pull_none>;
            };
      };
          ……
      }
```

配置设备树时，需要在 pinctrl 节点中配置引脚属性和功能。

rockchip,pins 属性说明：

- 第一个参数是 GPIO 控制器编号；
- 第二个参数是引脚组和序列号；
- 第三个参数是引脚功能；
- 最后一个是引脚属性。

例如：

将 gpio3_B3 配置为 GPIO 功能，无上下拉：

```
rockchip,pins = <3 RK_PB3 RK_FUNC_GPIO &pcfg_pull_none>;
```

7.2.2. 在内核驱动中配置 GPIO 方向

1. 配置 GPIO 为输入模式：

```
int ret;
ret = gpio_request_one(gpio_number, GPIOF_IN, "Linuxc-gpio");  if (!gpio_is_valid(gpio_
number)) {
pr_err("cary get gpio-nr failed\n"); return -EINVAL
}
```

2. 配置 GPIO 为输出模式，输出低电平：

```
int ret;
ret = gpio_request_one(gpio_number, GPIOF_OUT_INIT_LOW, "Linuxc-gpio"); if (ret) {
pr_err(“cary request gpio: %d failed\n”, gpio_number);
return -EBUSY;
}
```

3. 配置 GPIO 为输出模式，输出高电平：

```
int ret;
ret =    gpio_request_one(gpio_number, GPIOF_OUT_INIT_HIGH, " Linuxc-gpio");    if (ret) {
pr_err(“cary request gpio: %d failed\n”, gpio_number); return -EBUSY;
}
```

若返回值 *ret* 为 *0*，表示 GPIO 配置成功；若 *ret* 不为 *0*，表示 GPIO 配置失败。

7.2.3. 在内核驱动中获取 GPIO 电平

调用如下函数获取 GPIO 电平：

```
int ret;
ret = gpio_get_value(gpio_number);
```

7.2.4. 在内核驱动中修改 GPIO 输出电平

若当前 GPIO 被配置为输出模式，则可以修改 GPIO 电平，如下所示：

```
//修改当前 GPIO 输出低电平
gpio_set_value(gpio_number, 0);

//修改当前 GPIO 输出高电平
gpio_set_value(gpio_number, 1);
```

7.3. 用户空间 GPIO 操作示例

1. 通过 **adb shell** 命令操作 GPIO：

```
adb root && adb shell
    cat sys/kernel/debug/gpio                    //查看 GPIO 状态
    echo 65 > /sys/class/gpio/export             //申请 gpio2_A1
```

```
echo out > /sys/class/gpio/gpio65 /direction      //配置 gpio2_A1 为输出
echo 1 > /sys/class/gpio/gpio65 /value            //配置 gpio2_A1 输出高电平
echo 0 > /sys/class/gpio/gpio65 /value            //配置 gpio2_A1 输出低电平
cat /sys/class/gpio/gpio65 /value                 //读取 gpio2_A1 电平
echo in > /sys/class/gpio/gpio65 /direction       //配置 gpio2_A1 为输入
```

2. 在源码中控制 GPIO：

```
int ret, fd;

fd = open(“/sys/class/gpio/export”, O_WRONLY);
 if (fd < 0)
   return -1;

ret = write(fd, “21”, strlen(“21”));
if (ret < 0)
   return -1;
close(fd);

fd = open(“/sys/class/gpio/gpio21/direction”, O_RDWR);
 if (fd < 0)
   return -1;

ret = write(fd, “out”, strlen(“out”));
 if (ret < 0)
   return -1;
close(fd);

fd = open(“/sys/class/gpio/gpio21/value”, O_RDWR);
if (fd < 0)
   return -1;

ret = write(fd, “1”, strlen(“1”));
if (ret < 0)
   return -1;
 close(fd);
```

8 GMAC 功能

8.1. 概述

8.1.1. 以太网 PHY 芯片简介

GMAC 通常由 MAC 控制器和 PHY 构成。由于 QSM368ZP-WF 设备和 SG368Z 系列模块的 SoC 内置以太网 MAC 控制器，因此只需搭配一个以太网 PHY 芯片，即可实现以太网卡功能。不同厂家的 PHY 芯片有通用寄存器定义，通常情况下，配置这些通用寄存器即可让 PHY 正常工作。因此，理论上，如果不需要使用 PHY 厂家提供的自定义的寄存器配置实现一些个性化的功能，就无需修改 PHY 驱动。

备注

1. SG368Z 系列模块内置 2 个 GMAC（GMAC0 + GMAC1）控制器，最多支持 2 个 10/100/1000 Mbps 网口。
2. QSM368ZP-WF 设备内置 1 个 GMAC（GMAC1）控制器，支持 1 个 10/100/1000 Mbps 网口，PHY 芯片为 YT8531C。

8.1.2. 工作模式

以太网 PHY 芯片主要有四种工作模式，分别为：MII、RMII、GMII 和 RGMII。10/100 Mbps 以太网 PHY 与 MAC 之间的接口主要有 MII 和 RMII；10/100/1000 Mbps 以太网 PHY 与 MAC 之间的接口主要有 RGMII。

- MII（Medium Independent Interface，媒体独立接口）：支持 10 Mbps 和 100 Mbps 的操作，数据位宽为 4 位。
  - 在 100 Mbps 传输速率下，时钟频率为 25 MHz；
  - 在 10 Mbps 传输速率下，时钟频率为 2.5 MHz。
- RMII（Reduced MII，简化版 MII）：支持 10 Mbps 和 100 Mbps 的操作，数据位宽为 2 位。
  - 在 100 Mbps 传输速率下，时钟频率为 50 MHz；
  - 在 10 Mbps 传输速率下，时钟频率为 5 MHz。

- GMII（Gigabit MII，向下兼容 MII）：支持 10 Mbps、100 Mbps 和 1000 Mbps 的操作，数据位宽为 8 位。
  - 在 1000 Mbps 传输速率下，时钟频率为 125 MHz；
  - 在 100 Mbps 传输速率下，时钟频率为 25 MHz；
  - 在 10 Mbps 传输速率下，时钟频率为 2.5 MHz。
- RGMII（Reduced GMII，简化版 GMII）：支持 10 Mbps、100 Mbps 和 1000 Mbps 的操作，数据位宽为 4 位。
  - 在 1000 Mbps 传输速率下，时钟频率为 125 MHz，在时钟的上下沿同时采样数据；
  - 在 100 Mbps 传输速率下，时钟频率为 25 MHz，单个时钟沿采样；
  - 在 10 Mbps 传输速率下，时钟频率为 2.5 MHz，单个时钟沿采样。

在千兆以太网中，常用的接口为 GMII 和 RGMII 接口。其中，RGMII 接口的优势在于其同时适用于 10/100/1000 Mbps 通信速率，且占用的引脚数量较少。然而，RGMII 接口也存在一些缺点，即在设计 PCB 时需要注意时钟、控制和数据线的长度要相等，并且时序约束相对也更为严格。

8.2. 设备树配置

GMAC 的设备树配置文件为：

- Kernel 5.10 及之前：*kernel/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*
- Kernel 6.1：*kernel-6.1/arch/arm64/boot/dts/rockchip/rk3568-evb1-ddr4-v10.dtsi*

GMAC 的控制器节点默认配置如下图所示：

```
&gmac0 {
        phy-mode = "rgmii";
        clock_in_out = "output";

        snps,reset-gpio = <&gpio2 RK_PC6 GPIO_ACTIVE_HIGH>;
        snps,reset-active-high;
        /* Reset time is 20ms, 100ms for rtl8211f */
        snps,reset-delays-us = <0 20000 100000>;

        assigned-clocks = <&cru SCLK_GMAC0_RX_TX>, <&cru SCLK_GMAC0>;
        assigned-clock-parents = <&cru SCLK_GMAC0_RGMII_SPEED>, <&cru CLK_MAC0_2TOP>;
        assigned-clock-rates = <0>, <125000000>;

        pinctrl-names = "default";
        pinctrl-0 = <&gmac0_miim
                     &gmac0_tx_bus2
                     &gmac0_rx_bus2
                     &gmac0_rgmii_clk
                     &gmac0_rgmii_bus>;

        tx_delay = <0x3c>;
        rx_delay = <0x2f>;

        phy-handle = <&rgmii_phy0>;
        status = "disabled";
};
```

```
&gmac1 {
        phy-mode = "rgmii";
        clock_in_out = "output";

        snps,reset-gpio = <&gpio3 RK_PB0 GPIO_ACTIVE_HIGH>;
        snps,reset-active-high;
        /* Reset time is 20ms, 100ms for rtl8211f */
        snps,reset-delays-us = <0 20000 100000>;

        assigned-clocks = <&cru SCLK_GMAC1_RX_TX>, <&cru SCLK_GMAC1>;
        assigned-clock-parents = <&cru SCLK_GMAC1_RGMII_SPEED>, <&cru CLK_MAC1_2TOP>;
        assigned-clock-rates = <0>, <125000000>;

        pinctrl-names = "default";
        pinctrl-0 = <&gmac1m1_miim
                     &gmac1m1_tx_bus2
                     &gmac1m1_rx_bus2
                     &gmac1m1_rgmii_clk
                     &gmac1m1_rgmii_bus>;

        tx_delay = <0x4f>;
        rx_delay = <0x26>;

        phy-handle = <&rgmii_phy1>;
        status = "ok";
};
```

重要参数配置如下：

参数	描述
phy-mode	PHY 的工作模式。指定为 RGMII 或 RMII，*pinctrl-0* 字段必须与此字段相匹配。
clock_in_out	时钟输入、输出方向。 *input*：时钟由 PHY 输入给 MAC *output*：时钟由 MAC 输出给 PHY
snps,reset-gpio	用于复位 PHY 的 GPIO 配置。
snps,reset-active-high	复位 PHY 时，GPIO 处于高电平状态时有效。
snps,reset-delays-us	设置为 *0 20000 100000*，表示复位 PHY 前的延时为 0 毫秒，拉低维持的时间为 20 毫秒，拉高后延时 100 毫秒。
assigned-clocks	MAC 的时钟源。
assigned-clock-parents	MAC 父时钟，由*&ext_gmac* 提供。
pinctrl-0	设置为*&rgmii_pins* 或*&rmii_pins*，必须和 *phy-mode* 字段匹配。

8.3. 配置以太网常见问题排查

内核中以太网相关驱动默认打开，GMAC 驱动代码位于 *drivers/net/ethernet/stmicro/stmmac/*。

8.3.1. DMA 初始化失败

如果 GMAC 的驱动开机 log 上出现“DMA engine initialization failed”，可能是 GMAC 的工作时钟出现问题。需测量时钟引脚是否有时钟，时钟频率以及幅度等指标是否正常，主要确认以下几个方面：

1. 检查 iomux 配置是否正确、时钟脚寄存器值是否正确。
2. 检查时钟方向和配置是否与硬件匹配，参考***第 8.2 章***时钟设置。
3. 检查时钟树和 CRU 寄存器，确认时钟频率和时钟是否使能。

8.3.2. PHY 初始化失败

如果 GMAC 的驱动开机 log 上出现“No PHY found”或“Cannot attach to PHY”，表示找不到 PHY。找不到 PHY 时，主要排查以下几个原因：

1. 检查 MDC/MDIO iomux 寄存器值是否正确；

2. PHY 供电是否正常；

3. Reset IO 配置是否正确；

4. Reset IO 时序是否符合 PHY 数据表的要求，不同 PHY 的时序要求不一致，具体配置请参考 ***第 8.2 章***；

5. 测试 MDC/MDIO 波形是否异常，其中 MDC 时钟频率要求小于 2.5 MHz。

8.3.3. 连接问题

若出现连接问题，可以将 MDC/MDIO 与主芯片断开，直连至电脑进行排查，查看电脑端是否存在相同问题，以排除软件上的干扰，然后重点排查硬件上的问题。测试 TXN/P 以及 RXN/P 是否有 link 波形，如果不断出现 **link up/link down**，可能是 PHY 收到了错误的数据。排查方法如下：

1. 检查在 EEE 模式下，发送的波形 delay line 配置是否错误；

2. 检查供给 PHY 的时钟是否错误。

8.3.4. 数据不通

首先排查是否是 TX 或 RX 问题，还是二者都有问题。

8.3.4.1. TX

通过 **ifconfig -a** 查看 **eth0** 节点的 **TX packets** 的增加情况，若 **TX packets** 不断增加，表示 TX 数据在 GMAC 有发出数据。通过该节点可以查看网线是否已连接，如果 **TX packets** 为 **0**，则有可能网线没有连上；**carrier** 为 **1** 表示 **link up**，为 **0** 表示 **link down**，如下图所示：

```
eth0      Link encap:Ethernet  HWaddr 16:21:8d:d9:67:0b  Driver rk_gmac-dwmac
          inet6 addr: fe80::c43d:3e5d:533:b7ea/64 Scope: Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 TX bytes:2848
          Interrupt:45
```

将设备与 PC 连在同一个局域网内，在设备上 **Ping** PC，同时在 PC 端通过抓包工具（如 Wireshark）查看是否抓取了设备发出的数据，若抓到数据，表示 TX 数据是通的。如果没有抓到，需确认 TX 在哪个链路位置上出现了异常。可以通过测试 GMAC 的 TX Clock 与 TX Data 的波形，排除是 MAC 还是 PHY 出现问题。

MAC 可以检查以下几个方面：

1. 检查 TX Clock/TX Data 的 iomux 配置是否正确；
2. 检查 TX Clock 时钟频率是否正确；
3. 检查在 RGMII 模式下，TX delay line 配置是否正确。

PHY 端也可以测试 PHY 的 TXN/P 信号以确认 PHY 是否有数据发出。不同 PHY 的配置可能不同，具体需要查看 PHY 的数据表。

8.3.4.2. RX

若经过***第 8.3.4.1 章***的排查后，确定不是 TX 问题，则需重点排查 RX。连上网线后，通过 **ifconfig -a** 查看 **eth0** 节点的 **RX packets** 是否在不断增加，如果为 **0**，表示 GMAC RX 没有收到数据。如下图所示：

```
eth0      Link encap:Ethernet  HWaddr 16:21:8d:d9:67:0b  Driver rk_gmac-dwmac
          inet6 addr: fe80::c43d:3e5d:533:b7ea/64 Scope: Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:341 errors:0 dropped:0 overruns:0 frame:0
          TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:48928 TX bytes:3741
          Interrupt:355
```

可以通过测试 PHY 的 RXN/P，以及 GMAC 的 RX Clock/RX Data，来排除是 MAC 还是 PHY 出现问题。

MAC 可以检查以下几个方面：

1. 检查 RX Clock/RX Data 的 iomux 配置是否正确；
2. 检查 RX Clock 时钟频率是否正确；
3. 检查在 RGMII 模式下，TX delay line 配置是否正确；
4. 检查在 RGMII 模式下，RX delay line 配置是否正确。

假设 **RX packets** 为 **0**，**TX packets** 不断增加，但以太网仍无法正常通讯，则有可能是以下原因：

1. 在 RMII 模式下，MAC 和 PHY 的参考时钟不一致；
2. PHY 模式配置不正确，例如硬件上配置成了 MII 模式。

9 附录 参考文档及术语缩写

表 5：参考文档

文档名称
[1] Quectel_SG368Z 系列_硬件设计手册
[2] Quectel_SG368Z_Series_GPIO_Configuration
[3] Quectel_QSM368ZP-WF_用户指导

表 6：术语缩写

缩写	英文全称	中文全称
ADB	Android Debug Bridge	安卓调试桥
ADC	Analog-to-Digital Converter	模数转换器
AP	Access Point	接入点
ASCII	American Standard Code for Information Interchange	美国信息交换标准码
CPHA	Clock Phase	时钟相位
CPOL	Clock Polarity	时钟极性
CRU	Clock and Reset Unit	时钟和复位单元
CTS	Clear To Send	清除发送
DMA	Direct Memory Access	直接内存访问
DTS	Device Tree Source	设备树源文件
DTSI	Device Tree Source Include	包含设备树源文件
DWC3	DesignWare USB 3.0 Controller	DesignWare USB 3.0 控制器

EEE	Energy-Efficient Ethernet	节能以太网
EHCI	Enhanced Host Controller Interface	增强型主机控制器接口
FIFO	First In First Out	先进先出
FIQ	Fast Interrupt Request	快速中断请求
GMAC	Gigabit Media Access Controller	千兆介质访问控制器
GMII	Gigabit MII	向下兼容 MII
GPIO	General-Purpose Input/Output	通用型输入/输出
I2C	Inter-Integrated Circuit	集成电路总线
ID	Mostly refers to Identifier in terms of software	软件中多数指“标识符”
IIO	Industrial I/O	工业输入、输出
I/O	Input/Output	输入、输出
IOMUX	I/O Multiplexing	输入/输出多路复用
IP	Internet Protocol	网络协议
IRQ	Interrupt Request	中断请求
LSB	Least Significant Bit	最低有效位
MAC	Medium Access Control	介质访问控制
MDC	Management Data Clock	管理数据时钟
MDIO	Management Data Input/Output	管理数据输入/输出
MII	Medium Independent Interface	媒体独立接口
MTP	Media Transfer Protocol	多媒体传输协议
OHCI	Open Host Controller Interface	开放主机控制器接口
OTG	On-The-Go	OTG 技术
PC	Personal Computer	个人电脑
PCB	Printed Circuit Board	印刷电路板
PHY	Physical	端口物理层

PLL	Phase-Locked Loop	锁相环
QSGMII	Quad Serial Gigabit Media Independent Interface	四路串行千兆位中介介面
RAM	Random Access Memory	随机存取存储器
RGMII	Reduced GMII	简化版 GMII
RNDIS	Remote Network Driver Interface Specification	远程网络驱动接口规范
RMII	Reduced MII	简化版 MII
RTS	Ready To Send	请求发送
RX	Receive	接收
SAR	Successive Approximation Register	逐次逼近寄存器
SATA	Serial ATA	串行 ATA
SCL	Serial Clock Line	时钟信号线
SCLK	Serial Clock	串行时钟
SPI	Serial Peripheral Interface	串行外设接口
TX	Transmit	发送
UART	Universal Asynchronous Receiver/Transmitter	通用异步收发传输器
USB	Universal Serial Bus	通用串行总线
xHCI	Extensible Host Controller Interface	可扩展主机控制器接口